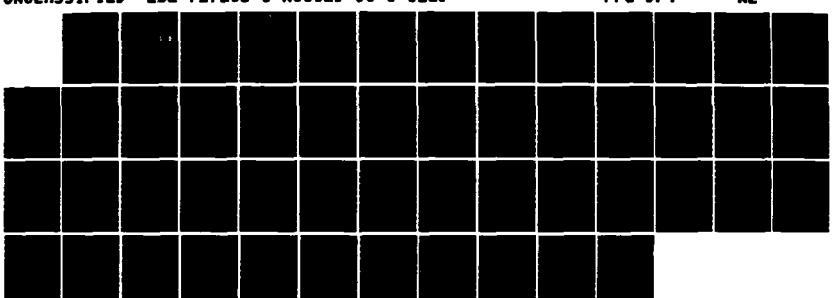
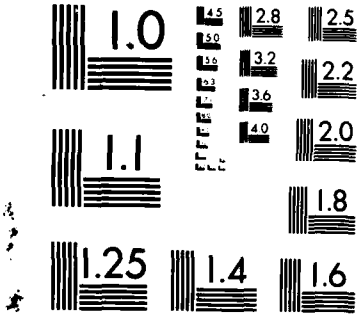
AD-A173 910 ON THE EQUIVALENCE BETWEEN TAPPED DELAY-LINE AND FFT PROCESSING IN ADAPTIVE ARRAYS(U) OHIO STATE UNIV COLUMBUS ELECTROSCIENCE LAB R T COMPTON JUN 86 1/1

UNCLASSIFIED ESL-717253-5 N00019-85-C-0119 F/G 9/4 NL





(13)

OSU

The Ohio State University

ON THE EQUIVALENCE BETWEEN TAPPED DELAY-LINE AND FFT PROCESSING IN ADAPTIVE ARRAYS

R. T. Compton, Jr.

The Ohio State University

ElectroScience Laboratory

Department of Electrical Engineering
Columbus, Ohio 43212

Final Technical Report 717253-5
Contract N00019-85-C-0119
June 1986

Department of the Navy
Naval Air Systems Command
Washington, DC 20361

NOTICES

When Government drawings, specifications, or other data are used for any purpose other than in connection with a definitely related Government procurement operation, the United States Government thereby incurs no responsibility nor any obligation whatsoever, and the fact that the Government may have formulated, furnished, or in any way supplied the said drawings, specifications, or other data, is not to be regarded by implication or otherwise as in any manner licensing the holder or any other person or corporation, or conveying any rights or permission to manufacture, use, or sell any patented invention that may in any way be related thereto.

50272-101

AD-A173 910

REPORT DOCUMENTATION PAGE	1. REPORT NO.	2.	3. Recipient's Accession No

4. Title and Subtitle
On the Equivalence between Tapped Delay-Line and FFT Processing in Adaptive Arrays

5. Report Date
June 1986

6.

7. Author(s)
R.T. Compton, Jr.

8. Performing Organization Rept. No
717253-5

9. Performing Organization Name and Address
The Ohio State University ElectroScience Laboratory
1320 Kinnear Rad
Columbus, Ohio 43212

10. Project/Task/Work Unit No

11. Contract(C) or Grant(G) No
(C) N00019-85-C-0119
(G)

12. Sponsoring Organization Name and Address
Department of the Navy
Naval Air Systems Command
Washington, DC 20361

13. Type of Report & Period Covered
Final Technical

14.

15. Supplementary Notes

16. Abstract (Limit: 200 words)

This report discusses the relationship between FFT processing and tapped delay-line processing in adaptive arrays. It is shown that the output SINR obtained from an adaptive array with FFTs behind the elements is identical to that of an equivalent adaptive array with tapped delay-line processing. The equivalent tapped delay-line array has the same number of taps in each delay-line as the number of samples used in the FFTs, and has a delay between taps equal to the delay between samples in the FFTs.

The main conclusion of this work is that FFT processing in and of itself does not offer any improvement in array bandwidth performance. The bandwidth performance of an adaptive array can be improved by using several time delayed samples of the signal from each element in the adaptive processor, rather than just a single sample. However, no further improvement results from taking FFTs of these sampled signals. The same bandwidth performance is obtained by simply weighting and combining the time domain samples directly.

17. Document Analysis a. Descriptors

b. Identifiers/Open-Ended Terms

c. COSATI Field/Group

18. Availability Statement
A. Approved for public release; distribution is unlimited.

19. Security Class (This Report)
Unclassified

21. No of Pages
49

20. Security Class (This Page)
Unclassified

22. Price

(See ANSI-Z39.18) See Instructions on Reverse

OPTIONAL FORM 272 (4-77)
(Formerly NTIS-35)
Department of Commerce

Contents

List of Figures

1. INTRODUCTION

An important problem with adaptive arrays [1,2] is that their performance deteriorates with interference bandwidth. The wider the bandwidth of an interference signal, the more difficult it is for an adaptive array to null it [3,4,5,6,7].

Figure 1.1, from [7], illustrates this problem. It shows the output SINR (desired signal-to-interference-plus-noise ratio) achieved by a two element adaptive array when an interference signal arrives from angle θ_i. The figure assumes two isotropic elements a half-wavelength apart, a desired signal with a 0 dB SNR (signal-to-noise ratio per element) arriving from broadside ($\theta_d = 0°$), and interference with a 40 dB INR (interference-to-noise ratio per element). The SINR is shown for several values of B, where B is the relative bandwidth, i.e., the ratio of the absolute bandwidth to the center frequency[1]. As may be seen, for $B = 0.02$ the output SINR has dropped about 3 dB below its value for $B = 0$. Larger bandwidths cause increasingly more degradation.

When the bandwidth performance of an adaptive array is inadequate, three methods exist for improving its performance:

1. Using more elements in the array,

2. Using tapped delay-lines behind the elements, and

3. Using FFTs (Fast Fourier Transforms) behind the elements.

Let us compare the performance obtained with each of these approaches.

[1] Figure 1 and subsequent figures in this Introduction have been computed as described in [7], and all notation here is identical to that in [7].

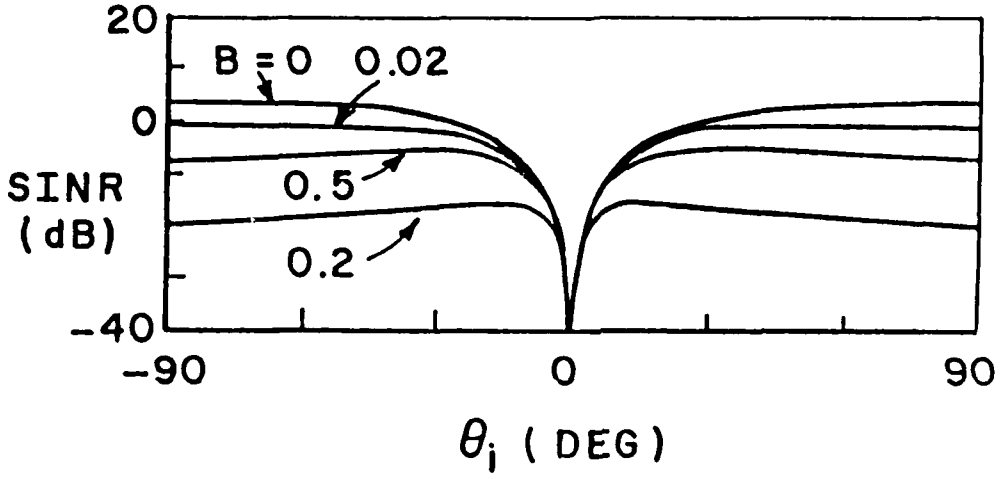
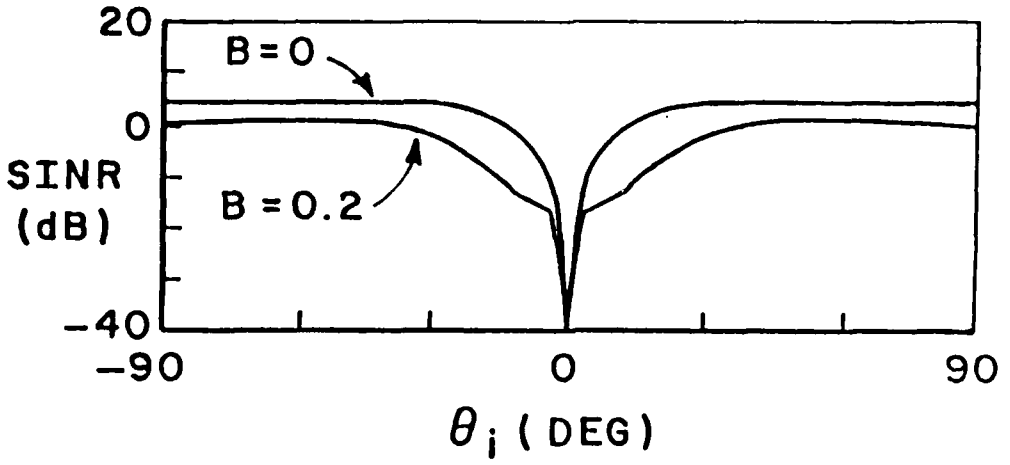


Figure 1.1: SINR versus θ_i
2-element array, one weight per element
$\theta_d = 0°$, SNR=0 dB, INR=40 dB

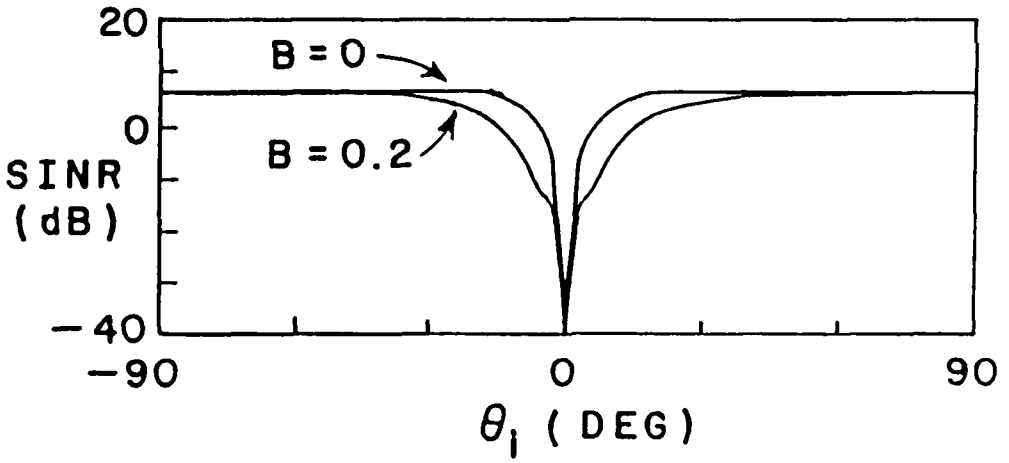
First, suppose we add more elements to the array. Figure 1.2 shows the SINR when the array has 3, 5, 10 or 20 elements, instead of 2 as in Figure 1.1. (In each case the array elements lie along a straight line with a half wavelength spacing between elements. All other parameters are the same as in Figure 1.1.) For each array size, the SINR is shown for $B = 0$ and $B = 0.2$. As may be seen from Figures 1.1 and 1.2, adding elements does improve the SINR. However, it is interesting that no matter how many elements are used, there is always a region for θ_i near θ_d where the SINR for $B = 0.2$ is poorer than for $B = 0$.

The second way to improve bandwidth performance is to use a tapped delay-line behind each element. (Figures 1.1 and 1.2 assume a single complex weight behind each element.) Figure 1.3 shows a two-element array with a 2-tap delay line behind each element. Figure 1.4 shows the output SINR vs. θ_i achieved by this array when $B = 0.2$ and when each delay is a quarter wavelength long at the carrier frequency. (The other parameters are the same as in Figure 1.) Comparing Figure 1.4 with Figure 1.1 shows that the delay lines behind the elements have fully overcome the bandwidth degradation. The performance for $B = 0.2$ in Figure 1.4 is just as good as that for $B = 0$ in Figure 1.1.

The third method that has been proposed for improving bandwidth performance is to use an FFT (Fast Fourier Transform) behind each element of the array, with a separate weight on each frequency bin. This approach is shown in Figure 1.5. At the output of each element, an A/D converter takes samples of the received signal. When K samples are available from each element, these samples are transformed with an FFT. The FFT produces K frequency domain samples of each element output. These frequency domain samples are each multiplied by a weight and then

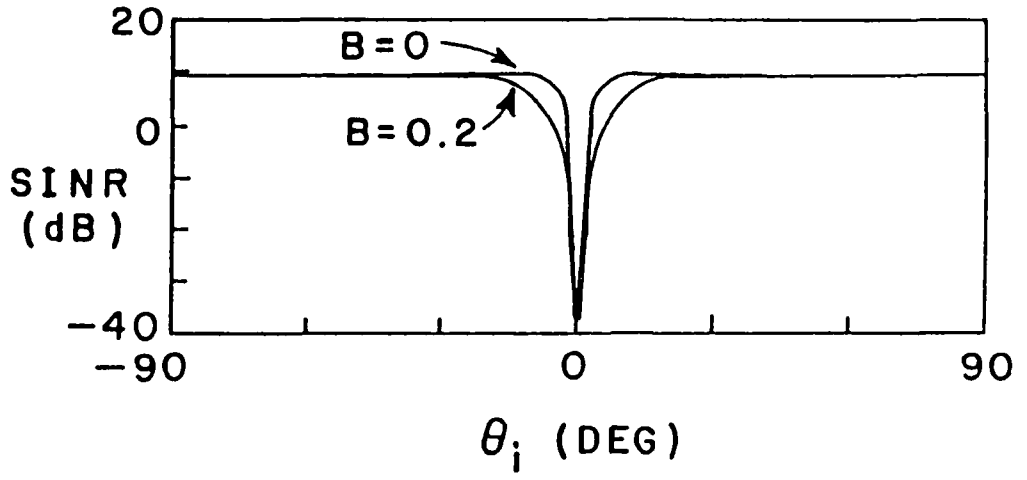


(a) 3 elements

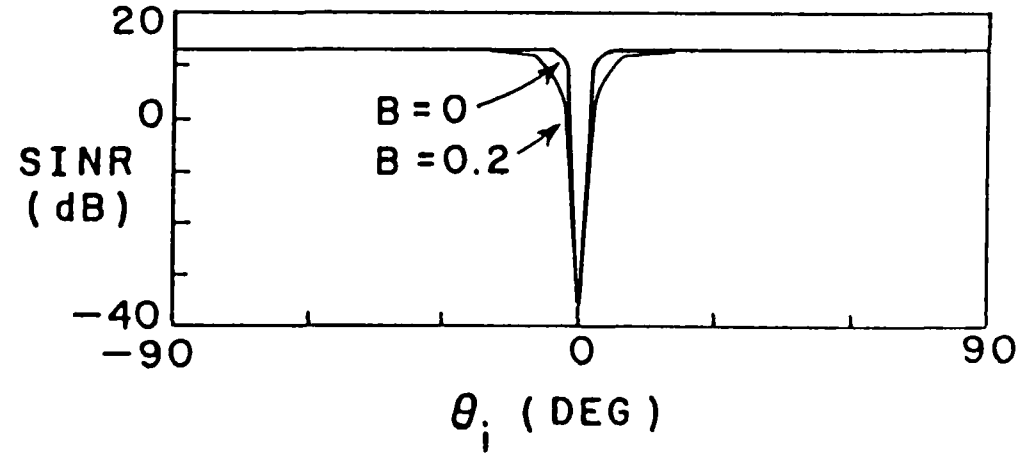


(b) 5 elements

Figure 1.2: SINR versus θ_i
One weight per element
$\theta_d = 0°$, SNR=0 dB. INR=40 dB



(c) 10 elements



(d) 20 elements

Figure 1.2: SINR versus θ_i
One weight per element
$\theta_d = 0°$, SNR=0 dB, INR=40 dB

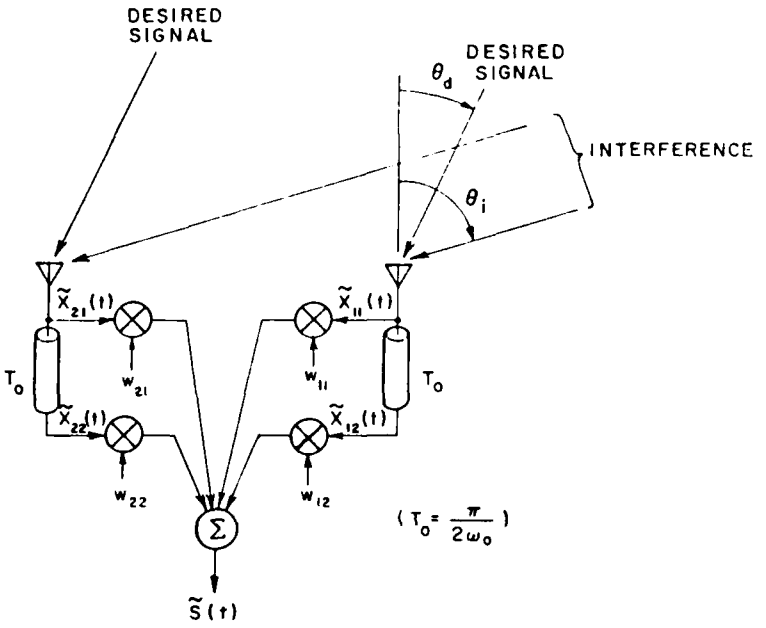


Figure 1.3: A 2-element array with 2-tap delay lines

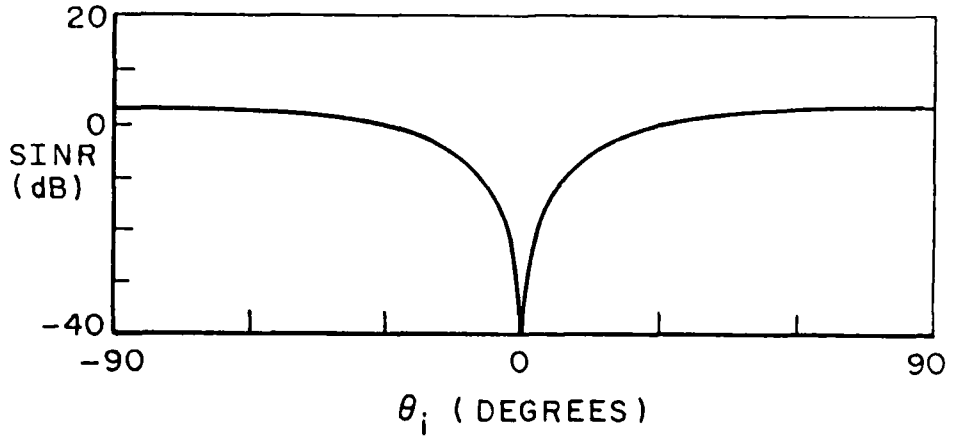


Figure 1.4: SINR versus θ_i
2-element array, 2-taps per element
$\theta_d = 0°$, SNR = 0 dB, INR = 40 dB

added to the corresponding frequency bins from other elements. Finally, the inverse FFT (IFFT) is taken of the frequency domain samples to obtain the time-domain array output samples.

The use of FFTs behind the elements has a certain intuitive appeal as a method of improving array bandwidth performance. In effect, the FFT divides the signal bandwidth into smaller subbands. (For this reason, this technique is sometimes called *frequency subbanding.*) With a separate weight on each frequency bin, the array can compensate differently for each frequency subband.

Unfortunately, calculations of array output SINR for FFT processing often indicate poorer performance than that for tapped delay lines. Figure 1.6 shows a typical set of results. It shows the output SINR from the same two-element array as in Figure 1.1, but with a K-point FFT behind each element. The curves are computed for the same bandwidth of $B = 0.2$ and for a sampling interval that makes one period of the FFT frequency response equal to the signal bandwidth. The SINR is shown for $K = 2, 4, 8$ and 16 samples in the FFTs. As may be seen, the performance does improve with K, but even for $K = 16$ it is not as good as the performance for tapped delay lines with only two taps, as seen in Figure 1.3. Thus, in spite of the intuitive appeal of FFTs, their performance can be disappointing.

The present study was done in an effort to understand the relationship between tapped delay-line and FFT processing in adaptive arrays. The study was motivated by the fact that there seemed to be no apparent reason why FFT performance should be poorer than tapped delay-line performance.

In this report we show that inserting an FFT between the delay-line taps and the weights in an adaptive array in fact has no effect on the output SINR. The

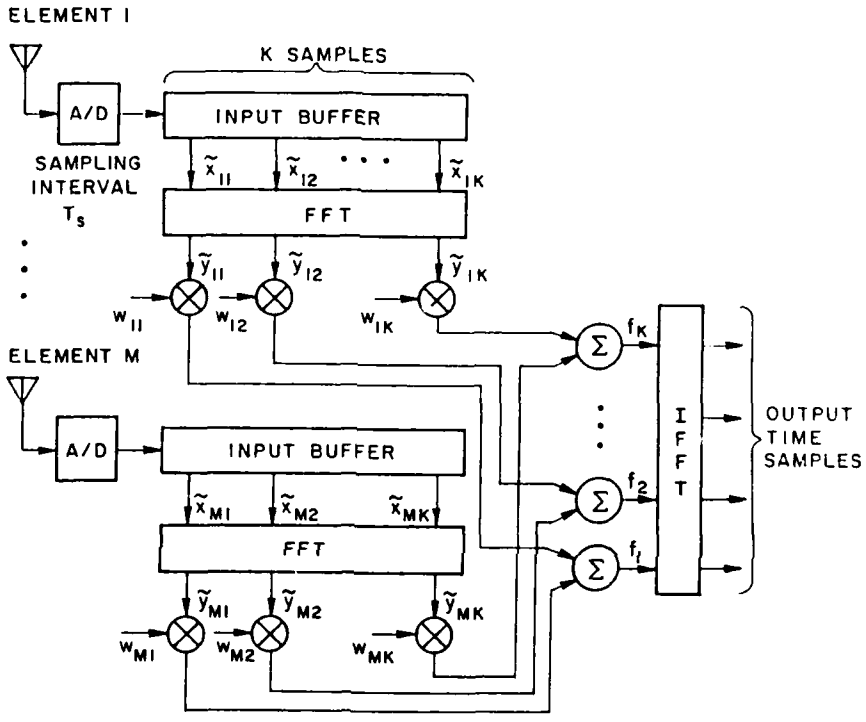


Figure 1.5: An array with FFT processing

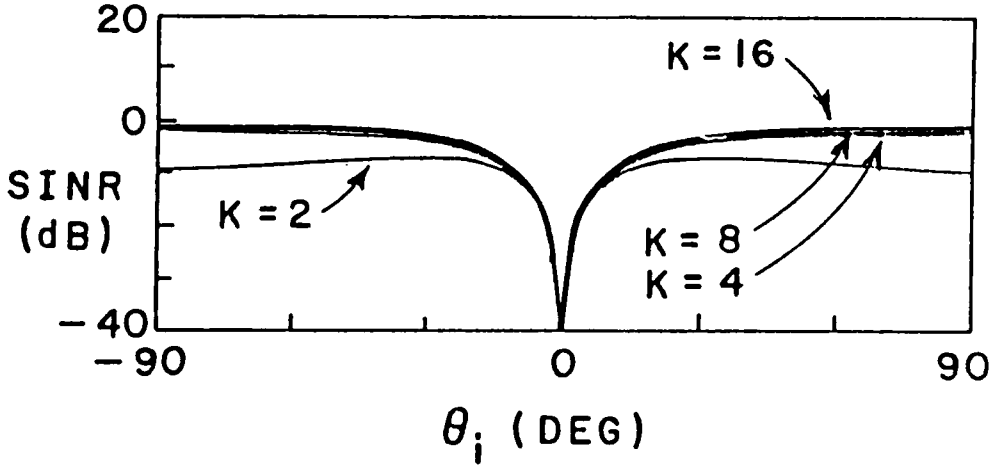


Figure 1.6: SINR versus θ_i
2-element array, K-point FFTs
$B = 0.2$, $\theta_d = 0°$, SNR=0 dB, INR=40 dB

difference in performance between the two approaches noted above is simply due to the different time delays between samples or different numbers of samples used in each case. When a tapped delay-line array and an FFT array use the same time between samples and the same number of samples, their performance is identical.

We proceed as follows. In Section 2, we first prove that inserting any linear, invertible transformation between the delay-line taps and the weights in an adaptive array has no effect on the array output SINR. We prove this theorem for both the LMS array and the Applebaum array. (For the Applebaum array, the proof holds only if the steering vector is also transformed in the appropriate way.) Next, in Section 3, we show that FFTs in an array simply provide a linear invertible transformation of the signals. Taken together, these two results show that FFT processing is entirely equivalent to tapped delay-line processing. Then, in Section 4, we discuss how FFT processing affects the array weights and the covariance matrix eigenvalue spread. We also discuss how tapped delay-line and FFT parameters are often chosen, and point out how these choices result in performance differences between the two methods. Finally, Section 5 contains the conclusions.

2. A SIMPLE THEOREM

In this section we present a simple theorem about adaptive arrays with tapped delay line processing. We show that inserting an arbitrary invertible linear transformation between the delay line taps and the adaptive weights has no effect on either the output signal or the output SINR. We prove this result for LMS arrays in Part 2.1 and for Applebaum arrays in Part 2.2. For Applebaum arrays, the steering vector must be transformed in the proper way along with the signals for the theorem to hold. We use this theorem in the next section to show that FFT processing is equivalent to tapped delay-line processing.

2.1 The LMS Array

Consider an adaptive array with M elements, as shown in Figure 2.1. Assume each element is followed by a tapped delay-line with K taps and a delay of T_0 seconds between taps. The output of the first tap behind each element is the element signal itself, with no delay. Let $\tilde{x}_{mk}(t)$ denote the (analytic) signal from element m at tap k. Then $\tilde{x}_{m1}(t)$ is the signal received on element m, and

$$\tilde{x}_{mk}(t) = \tilde{x}_{m1}(t - [k-1]T_0). \tag{2.1}$$

Let us suppose the $\tilde{x}_{mk}(t)$ are used as inputs to an LMS adaptive array processor [1]. This processor multiplies each $\tilde{x}_{mk}(t)$ by a complex weight w_{mk} and then adds the weighted signals to produce the array output $\tilde{s}_x(t)$, as shown in Figure 2.1. The processor uses LMS feedback loops [1] or an equivalent technique [8] to set the weights to their optimal values. The LMS optimal weights minimize the mean-square difference between the array output $\tilde{s}_x(t)$ and a reference signal $\tilde{r}(t)$, as

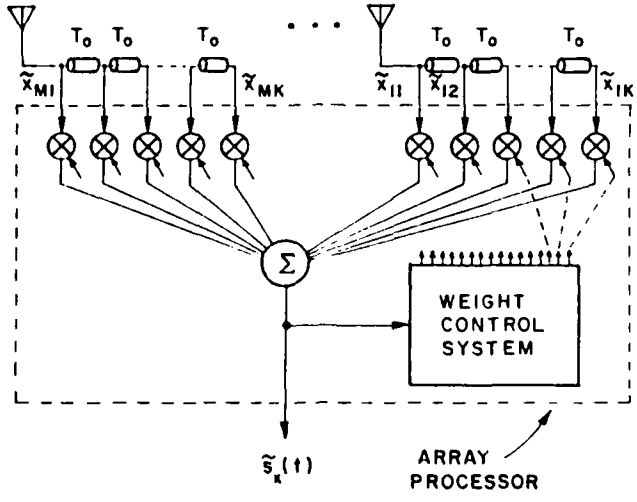


Figure 2.1: An M-element Array

described in [1]. Moreover, as long as the reference signal is correlated with the desired signal and uncorrelated with interference, the optimal weights also maximize SINR at the array output [9].

For a given set of tap signals $\tilde{x}_{mk}(t)$, the optimal weights may be calculated as follows. Let $X_m(t)$ and W_m (with $1 \leq m \leq M$) be column vectors containing the signals and weights at the K taps behind element m, i.e.,

$$X_m(t) = [\tilde{x}_{m1}(t),\ \tilde{x}_{m2}(t),\ \ldots,\ \tilde{x}_{mK}(t)]^T, \tag{2.2}$$

and

$$W_m = [w_{m1},\ w_{m2},\ \ldots,\ w_{mK}]^T. \tag{2.3}$$

(Superscript T denotes transpose.) We refer to $X_m(t)$ as the *element* signal vector and to W_m as the *element* weight vector. Then let $X(t)$ and W be the total signal

and weight vectors for the entire array,

$$X(t) = \begin{bmatrix} X_1(t) \\ - - \\ X_2(t) \\ - - \\ \vdots \\ - - \\ X_M(t) \end{bmatrix}, \tag{2.4}$$

and

$$W = \begin{bmatrix} W_1 \\ - - \\ W_2 \\ - - \\ \vdots \\ - - \\ W_M \end{bmatrix}, \tag{2.5}$$

where we use a partitioned vector notation. The optimal weight vector in the array is then given by [1,2]

$$W = \Phi_x^{-1} S_x, \tag{2.6}$$

where Φ_x is the signal covariance matrix,

$$\Phi_x = E[X^* X^T], \tag{2.7}$$

and S_x is the reference correlation vector.

$$S_x = E[X^* \tilde{r}(t)]. \tag{2.8}$$

In these equations, $*$ denotes complex conjugate and $\tilde{r}(t)$ is the reference signal [1]. We assume the signals $\tilde{x}_{mk}(t)$ as well as $\tilde{r}(t)$ are all jointly stationary random processes, so Φ_x and S_x do not depend on t.

With the weight vector W given by (2.6), the array output signal is

$$\tilde{s}_x(t) = X^T(t)W = X^T(t)\Phi_x^{-1}S_x. \tag{2.9}$$

Now consider the following alternative situation. Suppose that, instead of using the signals $\tilde{x}_{mk}(t)$ as inputs to the processor, we first combine them in some manner to produce a new set of signals $\tilde{y}_{mk}(t)$, where $1 \leq m \leq M$ and $1 \leq k \leq K$. Specifically, suppose Y_m is the K-component vector

$$Y_m = [\tilde{y}_{m1}(t),\ \tilde{y}_{m2}(t),\ \ldots,\ \tilde{y}_{mK}(t)]^T, \tag{2.10}$$

and Y is the MK-component vector,

$$Y(t) = \begin{bmatrix} Y_1 \\ - - \\ Y_2 \\ - - \\ \vdots \\ - - \\ Y_M \end{bmatrix} \tag{2.11}$$

Then let us assume that

$$Y(t) = TX(t), \tag{2.12}$$

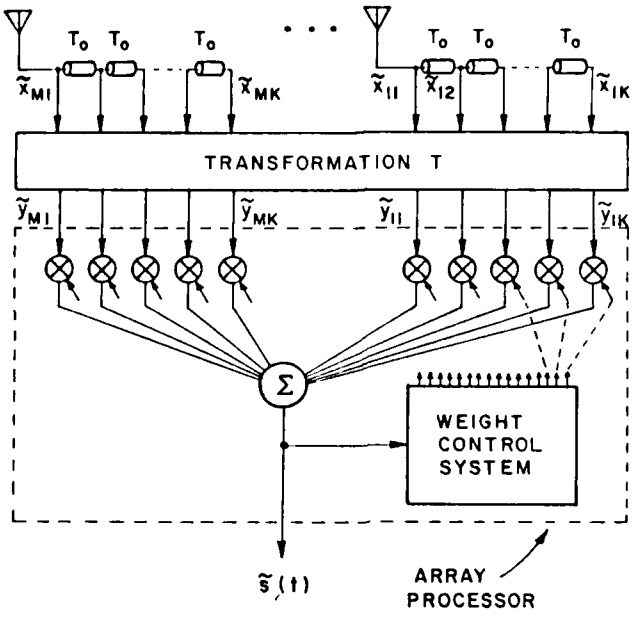


Figure 2.2: Tapped Delay-Line Array with Transformation T

where T is an $MK \times MK$ matrix of constants. Thus, each $\tilde{y}_{mk}(t)$ is a linear combination of the $\tilde{x}_{mk}(t)$. Now let us use the $\tilde{y}_{mk}(t)$ as inputs to the same LMS processor as before. Figure 2.2 shows the new arrangement with transformation T between the $\tilde{x}_{mk}(t)$ and the $\tilde{y}_{mk}(t)$. For this case, we denote the mk^{th} array weight by u_{mk}, to distinguish it from w_{mk} in Figure 2.1.

With the signals $\tilde{y}_{mk}(t)$ as inputs, the LMS processor will produce optimal weights given by

$$U = \Phi_y^{-1} S_y, \tag{2.13}$$

where U is the new weight vector,

$$U = [u_{11},\ u_{12},\ \ldots,\ u_{21},\ \ldots,\ u_{MK}]^T. \tag{2.14}$$

Φ_y is the covariance matrix associated with the signals $\tilde{y}_{mk}(t)$,

$$\Phi_y = E[Y^*(t)Y^T(t)], \tag{2.15}$$

and S_y is the reference correlation vector for the signals $\tilde{y}_{mk}(t)$,

$$S_y = E[Y^*(t)\tilde{r}(t)]. \tag{2.16}$$

$\tilde{r}(t)$ is the same reference signal as in (2.8). The array output signal for this case, $\tilde{s}_y(t)$, is

$$\tilde{s}_y(t) = Y^T(t)U = Y^T(t)\Phi_y^{-1}S_y. \tag{2.17}$$

Now it is easy to show that if T is invertible, $\tilde{s}_y(t)$ and $\tilde{s}_x(t)$ are in fact identical signals. Substituting (2.12) into (2.15) and (2.16), and using (2.7) and (2.8), we find

$$\Phi_y = E[Y^*(t)Y^T(t)] = E[T^*X^*(t)X^T(t)T^T] = T^*\Phi_x T^T, \tag{2.18}$$

and

$$S_y = E[Y^*(t)\tilde{r}(t)] = E[T^*X^*(t)\tilde{r}(t)] = T^*S_x. \tag{2.19}$$

If T is invertible (i.e., nonsingular), $\tilde{s}_y(t)$ in (2.17) reduces to

$$\begin{aligned}
\tilde{s}_y(t) &= Y^T(t)\Phi_y^{-1}S_y \\
&= X^T(t)T^T[T^*\Phi_x T^T]^{-1}T^*S_x \\
&= X^T(t)T^T[T^T]^{-1}\Phi_x^{-1}[T^*]^{-1}T^*S_x \\
&= X^T(t)\Phi_x^{-1}S_x \\
&= \tilde{s}_x(t).
\end{aligned} \tag{2.20}$$

Furthermore, the output SINR is the same for the two arrays. Substituting (2.18) and (2.19) into (2.13) shows that the weight vectors U and W are related by

$$\begin{aligned} U &= [T^* \Phi_x T^T]^{-1} T^* S_x \\ &= [T^T]^{-1} \Phi_x^{-1} [T^*]^{-1} T^* S_x \\ &= [T^T]^{-1} \Phi_x^{-1} S_x \\ &= [T^T]^{-1} W. \end{aligned} \tag{2.21}$$

Consider, for example, the output desired signal. Suppose $X_d(t)$ is the desired signal part of signal vector $X(t)$ and $Y_d(t)$ is the desired signal part of $Y(t)$. Then

$$Y_d(t) = T X_d(t). \tag{2.22}$$

The array output desired signal in Figure 2.1 is

$$\tilde{s}_{x_d}(t) = X_d^T(t) W, \tag{2.23}$$

whereas the output desired signal in Figure 2.2 is

$$\tilde{s}_{y_d}(t) = Y_d^T U. \tag{2.24}$$

However, using (2.21) and (2.22) in (2.24) gives

$$\tilde{s}_{y_d}(t) = [T X_d(t)]^T [T^T]^{-1} W = \tilde{s}_{x_d}(t). \tag{2.25}$$

Thus, the arrays in Figures 2.1 and 2.2 have identical output desired signals, and hence identical output desired signal powers. A similar argument shows that the output interference and thermal noise powers are also identical for the two arrays. From this it follows that the output SINR is the same in Figures 2.1 and 2.2.

Thus, inserting a linear transformation T between the elements and the processor, as in Figure 2.2, has no effect on the output signal or the output SINR. The only requirement for this result to hold is that the transformation be invertible.

2.2 The Applebaum Array

Now suppose the array processor in Figure 2.1 uses Applebaum control loops [2] or an equivalent technique [8] to set the weights w_{mk}. Applebaum control loops use a steering vector instead of a reference signal to maintain the array response in the desired signal direction. With Applebaum loops, the steady-state weight vector in the array will be [2]

$$W = \mu \Phi_x^{-1} V. \tag{2.26}$$

where μ is an arbitrary (nonzero) gain constant and V is the steering vector. The array output signal will be

$$\tilde{s}_x(t) = X^T W = \mu X^T(t) \Phi_x^{-1} V. \tag{2.27}$$

Now suppose a transformation T is inserted between the signals $\tilde{x}_{mk}(t)$ and the Applebaum processor, as in Figure 2.2. Let the processor now have a new steering vector Q. The steady-state weight vector in this case is

$$U = \mu \Phi_y^{-1} Q. \tag{2.28}$$

If (2.18) is substituted for Φ_y, (2.28) becomes

$$U = \mu [T^T]^{-1} \Phi_x^{-1} [T^*]^{-1} Q, \tag{2.29}$$

so the array output signal is

$$\begin{aligned} \tilde{s}_y(t) &= Y^T(t)U \\ &= \mu X^T(t)T^T[T^T]^{-1}\Phi_x^{-1}[T^*]^{-1}Q \\ &= \mu X^T(t)\Phi_x^{-1}[T^*]^{-1}Q. \end{aligned} \tag{2.30}$$

Comparing (2.30) with (2.27) shows that $\tilde{s}_x(t)$ and $\tilde{s}_y(t)$ will be the same if

$$V = [T^*]^{-1}Q, \tag{2.31}$$

i.e., if

$$Q = T^*V. \tag{2.32}$$

Thus, inserting the transformation T between the elements and the processor has no effect on the output signal (or, as with the LMS array, on the output SINR), if the steering vector is transformed according to (2.32).

We have now shown the major result of this section: placing an invertible transformation T between the delay-line taps and the adaptive processor has no effect on the output signal or the output SINR. This result holds for the LMS array and also for the Applebaum array if the steering vector is properly transformed.

It is important to note what this result says about array bandwidth performance. Since the transformation T has no effect on the output signal or SINR of an array, *it also can have no effect on the bandwidth performance of an array.* In other words, the theorem applies no matter what signals are present in the array. Whatever the signal bandwidths are, the array SINR will be the same with or without the transformation T. Thus, there is no invertible transformation T that one can

insert between the delay-line taps and the weights that will improve the bandwidth performance.

3. AN ARRAY WITH FFT PROCESSING

Now consider an array with FFT processing. Such an array was shown in Figure 1.5. An A/D converter behind each element samples the signal from that element every T_s seconds. The samples from each element are collected in the input buffer of a K-point FFT [10]. When K samples have been stored, the FFT is taken. This process generates K frequency domain samples from each element. These samples are multiplied by weights and then added in corresponding frequency bins to the weighted samples from other elements. The result is a set of K frequency domain samples of the array output. Finally, the inverse FFT (IFFT) is taken to obtain K time domain samples of the array output. This entire process is repeated every K samples.

The process described above is called block processing, since the input time samples are handled in blocks of K samples. After each block of K input samples is collected, a block of K array output time samples is generated by the inverse FFT. Each FFT cycle involves a block of K entirely new samples.

The FFT processing can also be done in a sliding window mode. In this case, the FFTs are recomputed after each new time sample, using always the most recent K time samples in each FFT input buffer. As we will see below, with this approach it is not necessary to do the inverse FFT to obtain the time domain array output. The time domain output is simply the sum of the weighted frequency domain samples.

Block processing has the advantage over sliding window processing that the FFTs need be computed only once per block, instead of once per sample. However, sliding window processing has the advantage that no inverse FFT is required to obtain the array output. In this section we shall consider both forms of processing.

We first define notation for the sampled element signals. To make the notation here consistent with that in Section 2, we denote the signal on the m^{th} element of the array by $\tilde{x}_{m1}(t)$. Let us concentrate on a particular set of K contiguous samples in each FFT input buffer in Figure 1.5. Suppose that, of these K samples, the most recent was taken at $t = t_0$. Then the latest sample of $\tilde{x}_{m1}(t)$ in the m^{th} FFT input buffer is $\tilde{x}_{m1}(t_o)$. The previous sample in that buffer is $\tilde{x}_{m1}(t_o - T_s)$, and the earliest sample is $\tilde{x}_{m1}(t_o - [K-1]T_s)$. If we define the signal $\tilde{x}_{mk}(t)$ as[1]

$$\tilde{x}_{mk}(t) = \tilde{x}_{m1}(t - [k-1]T_s), \tag{3.1}$$

we may write these samples as

$$\begin{aligned} \tilde{x}_{m1}(t_o) &= \tilde{x}_{m1}(t_o), \\ \tilde{x}_{m1}(t_o - T_s) &= \tilde{x}_{m2}(t_o), \\ &\cdots \\ \tilde{x}_{m1}(t_o - [K-1]T_s) &= \tilde{x}_{mK}(t_o). \end{aligned} \tag{3.2}$$

Now consider the FFT obtained from these samples. Let the K frequency domain samples produced by the FFT behind element m be denoted by $\tilde{y}_{m1}, \tilde{y}_{m2}, \ldots, \tilde{y}_{mK}$. These $\tilde{y}_{mn}$ are given by[2]

---

[1] Eq. (3.1) has the same form as Eq. (2.1) of Section 2, with T_0 replaced by T_s.

[2] It is common in the FFT literature [10] to denote the time domain samples by $x_0, x_1, \ldots, x_{K-1}$ and the frequency domain samples by $X_0, X_1, \ldots, X_{K-1}$. In this case the FFT is usually written

$$X_n = \sum_{k=0}^{K-1} x_k E_K^{kn}, \qquad 0 \le n \le K-1.$$

and the IFFT is

$$x_k = \frac{1}{K} \sum_{n=0}^{K-1} X_n E_K^{-kn}, \qquad 0 \le k \le K-1.$$

$$\tilde{y}_{mn} = \sum_{k=1}^{K} \tilde{x}_{mk}(t_0) E_K^{(k-1)(n-1)}, \qquad 1 \leq n \leq K, \tag{3.3}$$

where

$$E_K = e^{-j\frac{2\pi}{K}}. \tag{3.4}$$

The array processor multiplies each frequency domain sample $\tilde{y}_{mn}$ by a complex weight u_{mn}. We assume these weights are set to their optimum values, which maximize SINR at the array output. (The weights can be controlled with an LMS processor, an Applebaum processor, or any other equivalent processor.) The weighted samples are then combined, in one of two ways, to produce the array output. The method used to combine the samples depends on whether block processing or sliding window processing is used.

If block processing is used, the weighted frequency domain samples from each element are added in corresponding frequency bins, as shown in Figure 1.5. This step produces array output frequency domain samples $\tilde{f}_n$, given by

$$\tilde{f}_n = \sum_{m=1}^{M} u_{mn} \tilde{y}_{mn}. \tag{3.5}$$

Then the inverse FFT of the $\tilde{f}_n$ is taken to obtain the time domain samples of the array output. If we denote the array output signal by $\tilde{s}(t)$, and its samples by $\tilde{s}_k(t)$,

$$\tilde{s}_k(t) = \tilde{s}(t_0 - [k-1]T_s), \qquad 1 \leq k \leq K, \tag{3.6}$$

then the inverse FFT of the $\tilde{f}_n$ produces the following time samples of $\tilde{s}(t)$,

$$\tilde{s}_k(t_0) = \frac{1}{K} \sum_{n=1}^{K} \tilde{f}_n E_K^{-(k-1)(n-1)}, \qquad 1 \leq k \leq K. \tag{3.7}$$

However, to make our FFT notation correspond to that in Section 2, we instead write the FFT as in (3.3) and allow the indices k, n to vary from 1 to K.

In a practical array, the factor $1/K$ in (3.7) may be omitted. This factor is simply a gain constant in the array output signal, and it has no effect on the output SINR[3]. Hence we assume the array output is actually obtained from

$$\tilde{s}_k(t_0) = \sum_{n=1}^{K} \tilde{f}_n E_K^{-(k-1)(n-1)}, \qquad 1 \le k \le K. \tag{3.8}$$

Thus, with block processing, a block of K input samples is used in this manner to produce a block of K array output time samples. The entire process is repeated every KT_s seconds, using for each cycle an entirely new set of K samples from each element.

If sliding window processing is used, on the other hand, it is not necessary to perform the IFFT in (3.8). Note that the most recent sample of $\tilde{s}(t)$, $\tilde{s}(t_0)$, is given by (3.8) with $k = 1$,

$$\tilde{s}(t_0) = \tilde{s}_1(t_0) = \sum_{n=1}^{K} \tilde{f}_n. \tag{3.9}$$

Thus $\tilde{s}(t_0)$ is just the sum of the $\tilde{f}_n$. With sliding window processing, the other samples of $\tilde{s}(t)$ that could be found from (3.8) are not needed, because an entire FFT cycle is performed for each new input time sample. Successive samples of the array output are obtained simply by repeating (3.9) at each sample time. Hence the total array processing in this case is as shown in Figure 3.1.

Note that the optimal weights u_{mn} in the processor are the same regardless of whether block processing or sliding window processing is used. The optimal weights maximize SINR, which is the quantity,

[3] For an LMS processor, the weights are adjusted to make the array output match the reference signal. Omitting the factor $1/K$ just results in weights smaller by a factor $1/K$ than they would have been. For an Applebaum processor, the optimal weights contain an arbitrary gain constant anyway, such as μ in (2.26).

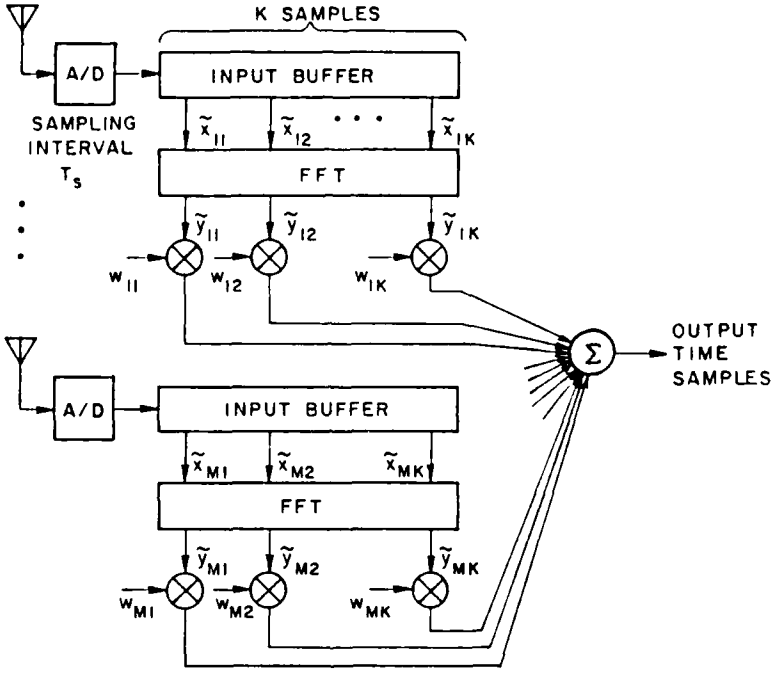


Figure 3.1: Sliding Window FFT Processing

$$\text{SINR} = \frac{P_d}{P_i + P_n}, \tag{3.10}$$

where P_d, P_i and P_n are the average desired, interference and thermal noise powers at the array output, respectively. If $\tilde{s}_d(t_j)$, $\tilde{s}_i(t_j)$ and $\tilde{s}_n(t_j)$ denote the desired, interference and thermal noise signals at the array output at a particular sample time t_j, then these powers are given by

$$P_d = E[|\tilde{s}_d(t_j)|^2], \tag{3.11}$$

$$P_i = E[|\tilde{s}_i(t_j)|^2], \tag{3.12}$$

and

$$P_n = E[|\tilde{s}_n(t_j)|^2]. \tag{3.13}$$

Because we assume the signals $\tilde{x}_{mk}(t)$ are stationary, the array output time sequence is a stationary random sequence. Hence each of the powers in (3.11)-(3.13) will have the same value regardless of which sample time it is computed at. Therefore the weights that maximize SINR at one sample time also maximize it at every other sample time.

To see that the optimal weights with sliding window processing are the same as those with block processing, it suffices to note that the array output for sliding window processing in (3.9) is the same as the array output for block processing in (3.8) when $k = 1$. Hence the weights that maximize the SINR at the $k = 1$ sample for block processing also maximize it for sliding window processing.

Since the optimal weights are the same for either type of processing, and since these weights produce the same output SINR in either case, we shall simplify the discussion below by considering only sliding window processing.

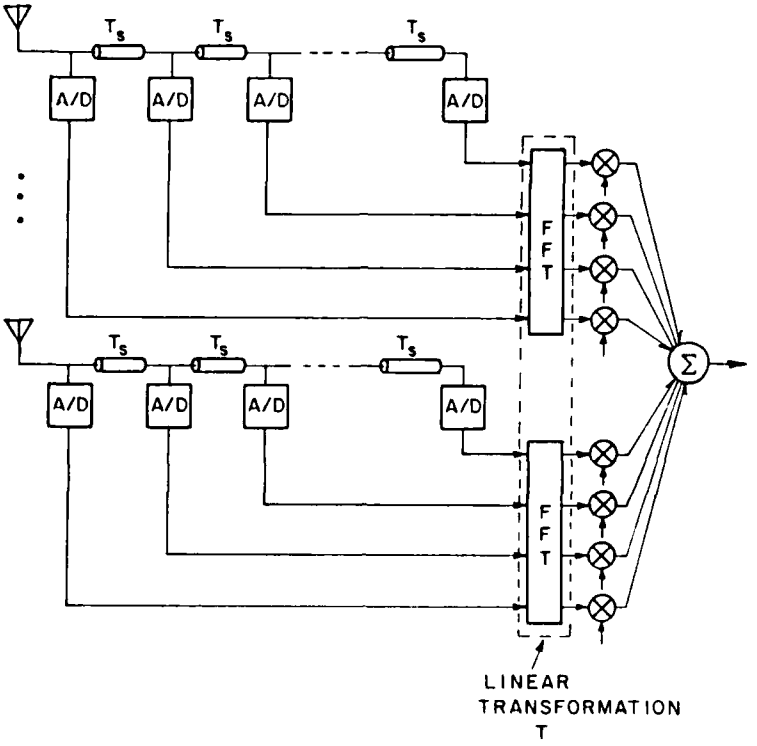


Figure 3.2: An Equivalent Tapped Delay-Line Array

Now let us consider the relationship between FFT processing and tapped delay-line processing. First, we note that Eq.(3.1) for the time samples in the FFT processor has the same form as Eq.(2.1) for the signals in a tapped delay-line processor, except that the intertap delay T_0 in (2.1) is replaced by the sampling time T_s in (3.1). Hence, for mathematical purposes, we may view the samples in the FFT processor as having been obtained from tapped delay-lines as shown in Figure 3.2. If the delay between taps in Figure 3.2 is T_s, and if every tap is sampled simultaneously at $t = t_0$, the same set of K samples will be obtained from the tapped delay-lines as from a single A/D converter behind each element.

Second, we note that the frequency domain samples $\tilde{y}_{mn}$ are each a linear combination of the input samples $\tilde{x}_{mk}(t_0)$. The linear combination is just the FFT in (3.3). Third, we have shown that the array output (for sliding window processing) is just the sum of the weighted frequency domain samples as in (3.9). Hence, the array with FFT processing is mathematically equivalent to an array with tapped delay-lines, followed by a linear transformation of the signals, followed by weighting and summing, as shown in Figure 3.2.

Moreover, note that the A/D converters at the delay-line taps in Figure 3.2 play no fundamental role in the operation of the array. The same array output samples would be produced by eliminating the A/D converters in Figure 3.2 and instead putting a single A/D converter at the array output as in Figure 3.3. The array is then an analog adaptive array with tapped delay-lines, followed by an A/D converter at the array output. The A/D converter in Figure 3.3 serves only to discretize the array output, but has no effect on the output SINR of the array.

The transformation between the $\tilde{x}_{mk}(t_0)$ and the $\tilde{y}_{mn}$ in (3.3) may be expressed in matrix form as in Section 2, of course. Let $X_m(t_0)$ be the element signal vector at time t_0,

$$X_m(t_0) = [\tilde{x}_{m1}(t_0),\ \tilde{x}_{m2}(t_0),\ \ldots,\ \tilde{x}_{mK}(t_0)]^T, \tag{3.14}$$

Then $X_m(t_0)$ contains the FFT input samples from element m used in (3.3). Also, let Y_m be a vector containing the frequency domain FFT samples from element m,

$$Y_m = [\tilde{y}_{m1},\ \tilde{y}_{m2}, \ldots,\ \tilde{y}_{mK}]^T. \tag{3.15}$$

Then Y_m and X_m are related by

$$Y_m = EX_m, \tag{3.16}$$

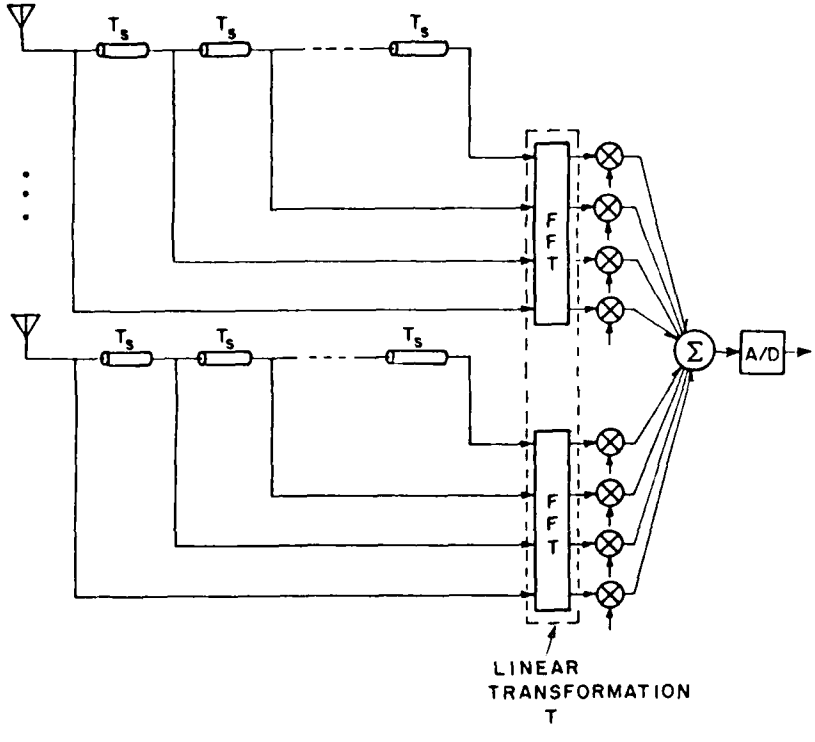


Figure 3.3: A Simpler Equivalent Tapped Delay-Line Array

where, from (3.3), E is the matrix

$$E = \begin{bmatrix} E_K^0 & E_K^0 & E_K^0 & \cdots & E_K^0 \\ E_K^0 & E_K^1 & E_K^2 & \cdots & E_K^{K-1} \\ E_K^0 & E_K^2 & E_K^4 & \cdots & E_K^{2(K-1)} \\ \vdots & \vdots & \vdots & & \vdots \\ E_K^0 & E_K^{K-1} & E_K^{2(K-1)} & \cdots & E_K^{(K-1)^2} \end{bmatrix}$$

$$= \begin{bmatrix} 1 & 1 & \cdots & 1 \\ 1 & e^{-j2\pi(\frac{1}{K})} & \cdots & e^{-j2\pi(\frac{K-1}{K})} \\ 1 & e^{-j2\pi(\frac{2}{K})} & \cdots & e^{-j2\pi(\frac{2(K-1)}{K})} \\ \vdots & \vdots & & \vdots \\ 1 & e^{-j2\pi(\frac{K-1}{K})} & \cdots & e^{-j2\pi(\frac{(K-1)^2}{K})} \end{bmatrix}. \qquad (3.17)$$

If $X(t_0)$ is the complete signal vector for the entire array,

$$X(t_0) = \begin{bmatrix} X_1(t_0) \\ - - \\ X_2(t_0) \\ - - \\ \vdots \\ - - \\ X_M(t_0) \end{bmatrix}, \qquad (3.18)$$

and Y is the vector containing all the frequency domain samples.

$$Y = \begin{bmatrix} Y_1 \\ - - \\ Y_2 \\ - - \\ \vdots \\ - - \\ Y_M \end{bmatrix}, \tag{3.19}$$

then Y and X are related by

$$Y = TX(t_0), \tag{3.20}$$

where

$$T = \left[\begin{array}{c|c|c|c} E & 0 & \cdots & 0 \\ - - & - - & - - - & - - \\ 0 & E & \cdots & 0 \\ - - & - - & - - - & - - \\ \vdots & \vdots & \ddots & \vdots \\ - - & - - & - - - & - - \\ 0 & 0 & \cdots & E \end{array}\right]. \tag{3.21}$$

Note that T is a block diagonal matrix. It has this form because each FFT involves the time samples from only one array element.

T is an invertible matrix, of course. The inverse of E in (3.17) is just the inverse FFT in matrix form,

$$E^{-1} = \frac{1}{K}\begin{bmatrix} E_K^0 & E_K^0 & \cdots & E_K^0 \\ E_K^0 & E_K^{-1} & \cdots & E_K^{-(K-1)} \\ E_K^0 & E_K^{-2} & \cdots & E_K^{-2(K-1)} \\ \vdots & \vdots & & \vdots \\ E_K^0 & E_K^{-(K-1)} & \cdots & E_K^{-(K-1)^2} \end{bmatrix}$$

$$= \frac{1}{K}E^*. \tag{3.22}$$

The inverse of T is then just

$$T^{-1} = \frac{1}{K}\left[\begin{array}{c|c|c|c} E^* & 0 & \cdots & 0 \\ \hline 0 & E^* & \cdots & 0 \\ \hline \vdots & \vdots & \ddots & \vdots \\ \hline 0 & 0 & \cdots & E^* \end{array}\right]$$

$$= \frac{1}{K}T^*. \tag{3.23}$$

We have thus shown that an array with FFT processing is mathematically equivalent to a tapped delay-line array with a linear invertible transformation between the taps and the weights. In the equivalent tapped delay-line array, the number of taps in each delay-line equals the number of time samples used in the FFTs, and the intertap delay T_0 equals the sampling time T_s. The theorem in Section 2 then

shows that the array with FFT processing will produce the same output SINR as the corresponding array with only the tapped delay-lines. The FFTs can be inserted or omitted with no change in performance.

An important conclusion that follows from this result is that FFT processing in and of itself does not offer any improvement in array bandwidth performance. The same bandwidth performance can be obtained simply by storing K samples of each element signal and then weighting these samples directly. Including the FFTs between the samples and the weights merely adds to the computational burden, but does nothing for the bandwidth performance.

4. ADDITIONAL COMMENTS ON FFT PROCESSING

In this section we discuss a few additional points of interest concerning FFT processing.

4.1 Optimal Weights With and Without FFT Processing

First, we consider how the optimal weights with FFT processing compare to those with tapped delay-line processing. Let U and W be the optimal weight vectors with and without the FFT transformation T in the array, respectively. Then, from (2.21), U and W are related by,

$$U = [T^T]^{-1}W. \tag{4.1}$$

However, the matrix T, given in (3.21), is symmetrical, because E in (3.17) is symmetrical. Hence (4.1) simplifies to

$$U = T^{-1}W. \tag{4.2}$$

Moreover, U and W may each be expressed in terms of *element* weight vectors U_m and W_m as in (2.5). Eq.(4.2) is then equivalent to

$$U_m = E^{-1}W_m, \qquad 1 \leq m \leq M. \tag{4.3}$$

Hence *the optimal weight vector behind each element with FFTs is just the inverse FFT of the optimal weight vector without FFTs.* Note that because (4.2) holds, the same array output signal is obtained with or without the FFTs,

$$Y^T U = [TX]^T[T^{-1}W] = X^T W, \tag{4.4}$$

as the theorem in Section 2 requires.

4.2 Covariance Matrix Eigenvalues

Next, we consider the eigenvalues of the covariance matrix seen by the adaptive array processor with and without the FFTs. These eigenvalues are of interest because they control the transient behavior or convergence properties of the algorithm used to adapt the weights [11]. Without the FFTs, the signal vector is X and the covariance matrix is

$$\Phi_x = E[X^* X^T]. \tag{4.5}$$

Suppose Φ_x has orthonormal eigenvectors e_{x_i} and eigenvalues λ_{x_i}[1],

$$\Phi_x e_{x_i} = \lambda_{x_i} e_{x_i}, \quad 1 \leq i \leq KM. \tag{4.6}$$

The eigenvectors e_{x_i} satisfy

$$e_{x_i}^\dagger e_{x_j} = \delta_{ij}, \quad 1 \leq i, j \leq KM. \tag{4.7}$$

where the superscript † denotes conjugate transpose and δ_{ij} is the Kronecker delta.

Now define new vectors

$$e_{y_i} = \sqrt{K} T^{-1} e_{x_i}, \quad 1 \leq i \leq KM. \tag{4.8}$$

These e_{y_i} also form an orthonormal set. From (4.8), we have

$$e_{y_i}^\dagger e_{y_j} = e_{x_i}^\dagger \sqrt{K} [T^{-1}]^\dagger \sqrt{K} T^{-1} e_{x_j}. \tag{4.9}$$

But because T is symmetrical ($T^T = T$) and $T^{-1} = \frac{1}{K} T^*$ (see (3.23)), we have

$$[T^{-1}]^\dagger = (\frac{1}{K} T^*)^\dagger = \frac{1}{K} T, \tag{4.10}$$

[1] Φ_x is a positive definite Hermitian matrix, so it has a complete set of eigenvectors and its eigenvalues are all real and positive.

so (4.9) reduces to

$$\begin{aligned} e_{y_i}^{\dagger} e_{y_j} &= K e_{x_i}^{\dagger} \frac{1}{K} T T^{-1} e_{x_j} \\ &= e_{x_i}^{\dagger} e_{x_j} \\ &= \delta_{ij}, \qquad 1 \le i, j \le KM. \end{aligned} \tag{4.11}$$

Now in (4.6) let us substitute $e_{x_i} = \frac{1}{\sqrt{K}} T e_{y_i}$ (the inverse of (4.8)) and multiply on the left by T^{-1}. We obtain

$$T^{-1} \Phi_x T e_{y_i} = \lambda_{x_i} e_{y_i}. \tag{4.12}$$

Then we replace T^{-1} by $\frac{1}{K} T^*$ and T by T^T,

$$[T^* \Phi_x T^T] e_{y_i} = K \lambda_{x_i} e_{y_i}. \tag{4.13}$$

Finally, from (2.18) we note that

$$T^* \Phi_x T^T = \Phi_y, \tag{4.14}$$

so (4.13) is just

$$\Phi_y e_{y_i} = K \lambda_{x_i} e_{y_i}, \quad 1 \le i \le KM. \tag{4.15}$$

Eq.(4.15) shows that e_{y_i} and $K\lambda_{x_i}$ are the i^{th} eigenvector and eigenvalue of Φ_y. Thus, each eigenvalue of Φ_y is simply K times the corresponding eigenvalue of Φ_x.

From this it follows that Φ_y and Φ_x have the same eigenvalue *spread.* (The eigenvalue spread of a matrix is the ratio of its largest to its smallest eigenvalue.) Hence typical problems caused by eigenvalue spread, such as long convergence times, roundoff errors in covariance matrix inversions, etc., will be the same with or without the FFTs.

4.3 Weight D· ..mic Range

Now we consider an issue of practical interest: how FFT processing affects the dynamic range of the weights. We may gain insight into this question as follows.

McClellan and Parks [12] have studied the eigenstructure of the FFT transformation matrix. From their results, it is easily shown that the matrix E in (3.17) has a complete set of orthonormal eigenvectors e_{E_j}, $1 \leq j \leq K$, and that every eigenvalue λ_{E_j} of E has one of the four values $\sqrt{K}$, $-\sqrt{K}$, $+j\sqrt{K}$, or $-j\sqrt{K}$. The multiplicity of each eigenvalue varies with K, the order of E.

From the eigenvectors and eigenvalues of E one can obtain the eigenvectors and eigenvalues of T in (3.21) in an obvious way. Each eigenvector e_{T_j} of T will have $(M-1)K$ components equal to zero and K components consisting of one eigenvector e_{E_j} of E. The eigenvalues λ_{T_j} of T will be the same as those of E but with multiplicities M times higher.

Suppose W is the optimum weight vector without FFTs and U the optimum weight vector with FFTs. Then from (2.21) we have

$$U = [T^T]^{-1}W = T^{-1}W. \tag{4.16}$$

The optimal weight vector W may be expressed in terms of its components along each of the eigenvectors of T,

$$W = \sum_{j=1}^{MK} \alpha_j e_{T_j}. \tag{4.17}$$

where each α_j is a scalar constant. Moreover, the matrix T^{-1} may be written in terms of the eigenvectors e_{T_j} and eigenvalues λ_{T_j} of T using the spectral decomposition formula,

$$T^{-1} = \sum_{j=1}^{MK} \frac{1}{\lambda_{T_j}} e_{T_j} e_{T_j}^{\dagger}, \tag{4.18}$$

Substituting (4.17) and (4.18) into (4.16) then gives

$$U = \sum_{j=1}^{MK} \frac{\alpha_j}{\lambda_{T_j}} e_{T_j}. \tag{4.19}$$

We cannot determine the magnitude of the components of U exactly without considering specific cases, because some λ_{T_j} are real and some are imaginary. But because $|\lambda_{T_j}| = \sqrt{K}$ for every j, (4.19) shows that the weights with FFTs will generally be smaller than the weights without FFTs by a factor of about $\frac{1}{\sqrt{K}}$.

4.4 Performance Differences between Tapped Delay-Line and FFT Processing

Finally, let us consider how the delay parameters are usually chosen in tapped delay-line and FFT processors. In the Introduction we noted that array performance can be poorer for FFT processing than for tapped delay-line processing. Figure 1.6, showing typical results with FFTs, was compared with Figure 1.4 for tapped delay-lines.

However, it is clear from the results of Sections 2 and 3 that the SINR performance of an array with FFTs must be identical to that of the equivalent tapped delay-line array. The equivalent tapped delay-line array has the same number of taps as the number of samples in the FFTs and has an intertap delay equal to the FFT sampling interval.

The performance difference noted in the Introduction is due entirely to the fact that typical comparisons have assumed different intertap delays or different numbers of taps for the two types of arrays. For a tapped delay-line processor, the intertap

delay is often assumed to be a quarter wavelength at the carrier frequency. For an FFT processor, the sampling time is usually chosen so the period of the FFT frequency response approximates the signal bandwidth. These two amounts of delay are usually very different.

Let us consider these time delays. First, suppose the signal carrier frequency is ω_0. The time delay required to produce a 90° phase shift at the carrier frequency (a quarter-wave delay) is then

$$T_{90^\circ} = \frac{\pi}{2\omega_0}. \tag{4.20}$$

In general, suppose we have

$$T_0 = rT_{90^\circ}, \tag{4.21}$$

where r is the number of quarter-wave delays in T_0. Although there is actually no fundamental reason to do so, with tapped delay-line processing it is common [4] to assume $r = 1$.[2]

Now consider the choice of T_s in an FFT processor. We may view the FFT in (3.3) as being equivalent to a filter bank. The input to the filter bank is $\tilde{x}_{m1}(t)$ and the outputs are $\tilde{y}_{m1}$, $\tilde{y}_{m2}, \ldots,$ $\tilde{y}_{mK}$. One filter produces the output $\tilde{y}_{m1}$, another produces the output $\tilde{y}_{m2}$, and so forth. Let us consider the transfer function of each of these filters.

Suppose the input signal $\tilde{x}_{m1}(t)$ is a sinusoid at frequency ω,

$$\tilde{x}_{m1}(t) = e^{j\omega t}. \tag{4.22}$$

Then, from (3.1),

[2] It is shown in [7] that any choice of r in the range $0 \leq r \leq \frac{1}{D}$ will work just as well.

$$\tilde{x}_{mk}(t) = e^{j\omega[t-(k-1)T_s]}. \tag{4.23}$$

For a specific n, the output signal $\tilde{y}_{mn}$ may be found by substituting (4.23) into (3.3),

$$\begin{aligned} \tilde{y}_{mn} &= \sum_{k=1}^{K} \tilde{x}_{mk}(t) e^{-j\frac{2\pi}{K}(k-1)(n-1)} \\ &= \sum_{k=1}^{K} e^{-j(k-1)[\omega T_s + \frac{2\pi}{K}(n-1)]} e^{j\omega t}. \end{aligned} \tag{4.24}$$

This may be written,

$$\tilde{y}_{mn} = H_n(\omega) e^{j\omega t}, \tag{4.25}$$

where $H_n(\omega)$, the n^{th} transfer function, is

$$\begin{aligned} H_n(\omega) &= \sum_{k=1}^{K} e^{-j(k-1)[\omega T_s + \frac{2\pi}{K}(n-1)]} \\ &= e^{-j\frac{K-1}{2}[\omega T_s + \frac{2\pi}{K}(n-1)]} \frac{sin\frac{K}{2}[\omega T_s + \frac{2\pi}{K}(n-1)]}{sin\frac{1}{2}[\omega T_s + \frac{2\pi}{K}(n-1)]}. \end{aligned} \tag{4.26}$$

In general, $H_n(\omega)$ is a periodic function of frequency with $\frac{sinK\omega}{sin\omega}$ peaks at frequencies

$$\omega = \frac{2\pi}{T_s}[i - \frac{n-1}{K}]. \qquad i = \ldots, -2, -1, 0, 1, 2, \ldots \tag{4.27}$$

For a given n, the peaks of $H_n(\omega)$ occur every $\frac{2\pi}{T_s}$ along the frequency axis. For adjacent n, the peaks are separated by $\frac{2\pi}{KT_s}$. Figure 4.1 shows a typical set of $H_1(\omega)$, ..., $H_K(\omega)$ over part of the frequency axis.

In studies of FFT processing, it is common to choose T_s so one complete set of K filter passbands approximately covers the signal bandwidth. (This choice seems sensible, since it divides the signal bandwidth into K subbands.) If the signal bandwidth is $\triangle\omega$, we set

$$\frac{2\pi}{T_s} = \triangle\omega, \tag{4.28}$$

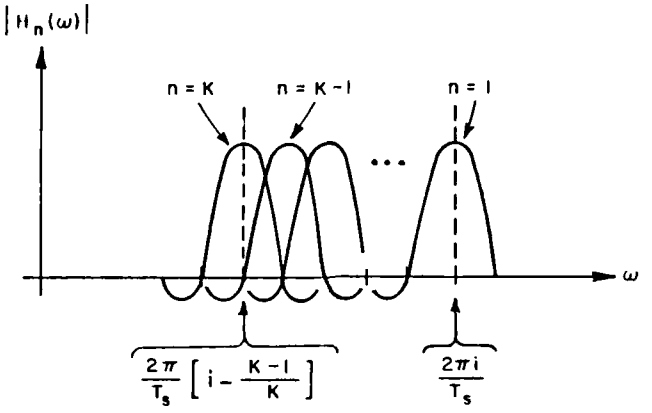


Figure 4.1: The Transfer Functions $H_1(\omega)$, ..., $H_K(\omega)$

or

$$T_s = \frac{2\pi}{\triangle\omega}. \tag{4.29}$$

However, (4.29) may be rearranged into the form

$$T_s = 4\,\frac{\pi}{2\omega_0}\,\frac{\omega_0}{\triangle\omega} = \frac{4}{B}T_{90^\circ}, \tag{4.30}$$

where B is the relative bandwidth of the signals,

$$B = \frac{\triangle\omega}{\omega_0}, \tag{4.31}$$

and T_{90° is given in (4.20). If, for example, the signal has a 1% relative bandwidth, then

$$T_s = 400T_{90^\circ}. \tag{4.32}$$

Note that this choice corresponds to $r = 400$ in (4.21), i.e., to an intertap delay of 400 quarter-waves in the equivalent tapped delay-line array!

In Figure 1.4 of the Introduction, which shows the SINR of a 2-element tapped delay-line array, there are two taps per element and one quarter-wave delay between taps. In Figure 1.6, which shows the SINR for a 2-element array with FFT processing, the sampling time has been selected according to (4.29). Thus, the tapped delay-line array in Figure 1.4 and the FFT array in Figure 1.6 are not equivalent. The difference in their performance is due to the difference between T_0 and T_s, as well as K.

An earlier report [7] discusses in detail how the number of taps and the amount of intertap delay affect the SINR performance of a two-element tapped delay-line array. In particular, it was shown there that setting T_s according to (4.29) (i.e., setting $r = 4/B$) makes r too large to obtain optimal SINR from the array. For optimal SINR, r must be in the range $0 < r < \frac{1}{B}$. Thus, although it seems intuitively sensible to choose T_s so the signal bandwidth is divided into K subbands, in fact this choice yields suboptimal SINR. Better performance will be obtained if T_s is chosen so the FFT period is at least four times the signal bandwidth. The reader is referred to [7] for further discussion of this question.

5. CONCLUSIONS

In this report we have shown that the SINR performance of an adaptive array with FFT processing is identical to that of an adaptive array with equivalent tapped delay-line processing. In the equivalent tapped delay-line processing, the number of taps in the delay-lines is equal to the number of samples used in the FFTs, and the delay between taps is equal to the delay between samples in the FFTs.

In Section 2, we showed that inserting a linear invertible transformation between the delay-line taps and the weights in a tapped delay-line array has no effect on the array output signal or SINR. Whatever changes are caused in the signals by the linear transformation are compensated for by corresponding changes in the weights. Then in Section 3 we showed that using FFTs behind each element is mathematically equivalent to using tapped delay-lines followed by a linear transformation of the signals. From the results of Sections 2 and 3, the main conclusion follows.

In Section 4, we also discussed the effects of FFTs on the optimal weights, the covariance matrix eigenvalues, and the dynamic range of the weights. Finally, the reasons for the performance differences between FFT processing and tapped delay-line processing noted in the Introduction were discussed.

The most important conclusion that follows from these results is that FFT processing in and of itself does not offer any improvement in array bandwidth performance. The same bandwidth performance will be obtained by simply storing K samples from each element signal and then weighting and combining these samples directly. Performing FFT calculations between the samples and the weights adds nothing.

6. REFERENCES

[1] B. Widrow, P. E. Mantey, L. J. Griffiths and B. B. Goode, "Adaptive Antenna Systems," Proc. IEEE, *55*, 12 (December 1967), 2143.

[2] S. P. Applebaum, "Adaptive Arrays," IEEE Trans. on Antennas and Propagation, *AP-24*, 5 (September 1976), 585.

[3] R. A. Monzingo and T. W. Miller, *Introduction to Adaptive Arrays*, John Wiley and Sons, New York, NY, 1980.

[4] W. E. Rodgers and R. T. Compton, Jr., "Adaptive Array Bandwidth with Tapped Delay-Line Processing," IEEE Trans. on Aerospace and Electronic Systems, *AES-15*, 1 (January 1979), 21.

[5] J. T. Mayhan, A. J. Simmons and W. C. Cummings, "Wideband Adaptive Antenna Nulling Using Tapped Delay Lines," IEEE Trans. on Antennas and Propagation, *AP-29*, 6 (November 1981), 923.

[6] W. D. White, "Wideband Interference Cancellation in Adaptive Sidelobe Cancellers," IEEE Trans. on Aerospace and Electronic Systems, *AES-19*, 6 (November 1983), 915.

[7] R. T. Compton, Jr., "The Bandwidth Performance of a Two-Element Adaptive Array with Tapped Delay-Line Processing," Quarterly Technical Report 717253-3, April 1986, ElectroScience Laboratory, Ohio State University, Columbus, OH 43212; prepared under Contract N00019-85-C-0119 for Naval Air Systems Command, Washington, DC 20361.

[8] I. S. Reed, J. D. Mallett and L. E. Brennan, "Rapid Convergence Rate in Adaptive Arrays," IEEE Trans. on Aerospace and Electronic Systems, *AES-10*, 6 (November 1974), 853.

[9] C. A. Baird and C. L. Zahm, "Performance Criteria for Narrowband Array Processing," 1971 IEEE Conference on Decision and Control, Miami Beach, FL, December 15-17, 1971.

[10] A. V. Oppenheim and R. W. Schafer, *Digital Signal Processing*, Prentice-Hall, Inc., Englewood Cliffs, NJ, 1975.

[11] R. T. Compton, Jr., *Adaptive Antennas - Concepts and Performance*, to be published by Prentice-Hall, Englewood Cliffs, NJ, 1987.

[12] J. H. McClellan and T. W. Parks, "Eigenvalue and Eigenvector Decomposition of the Discrete Fourier Transform," IEEE Trans. on Audio and Electroacoustics, *AU-20*, (March 1972), 66.

END

12-86

DTIC